SHARP APL *for* OS/390

Guide for APL Programmers

Version 23

SOLITON ASSOCIATES

Guide for APL Programmers

Version 23

SHARP APL is a registered trademark of Soliton Associates Limited.
All other products referred to in this document are trademarks or registered trademarks of their respective trademark holders.

© 2001 by Soliton Associates Limited — Printed in Canada
44 Victoria Street, Suite 2100, Toronto, Ontario, Canada M5C 1Y2

All rights reserved. No part of this work may be reproduced in any form or by any means without prior written permission of the copyright owner.

Soliton Associates Limited shall not be liable for errors contained herein or for incidental consequential damages in connection with the use of this material. The information contained within this document is subject to change without notice.

Publication code: MM-000-1503 (0102)

Contents

Preface

Introduction v
Who Should Read This Guide? v
Conventions vi
The APL Union Character Set vi
Related Publications vii
Contacting Soliton Associates vii

1. Introduction

Overview 1-1
Summary of Changes 1-1
Language Enhancements 1-1
Intrinsic Functions 1-1
Other Enhancements 1-2

2. Language Enhancements

Overview 2-1
Enhancements 2-1
Allow underbars in APL names 2-1
The Euro Currency symbol is now available 2-1
⎕*avm* Added 2-1
Changes 2-2
System Errors with Enclosed Arrays 2-2
⎕*io* 2-2
less 2-2

3. Intrinsic Functions

Overview 3-1
sapl . Group 3-1
Enhancements 3-2
Date Conversion Intrinsics 3-2

sapl.nextprime 3-3
sapl.fbrowmatch 3-3
Changes 3-4
sapl.rowmatch 3-4
sapl.translate 3-4

4. Other Enhancements
Enhancements 4-1
Case Insensitive Passwords 4-1
)cc Corrections 4-1
SVP 4-2

Preface

Introduction

SHARP APL for OS/390 Version 23 is a data base oriented release. It provides a number of enhancements to the SHARP APL product, including file system improvements and a speed up of interpreter components utilized by a typical data base application. These changes include modifications and additions to the base APL system, the SHARP APL utilities, and other related software products.

The *SHARP APL for OS/390 Guide for APL Programmers* describes the changes to SHARP APL since V22.2 as they relate to programmers.

This document, when used with the *SHARP APL for OS/390 Internal and Operation Changes* guide provides a summary of all changes to SHARP APL and related products distributed as V23. Detailed procedures for upgrading to V23 are provided in the *SHARP APL for OS/390 Upgrade Guide*.

Note: Rather than reproduce existing material, references to related SHARP APL publications are supplied where applicable.

Who Should Read This Guide?

All programmers, both professional programmers and occasional interactive users, using SHARP APL for OS/390 should read this guide.

Conventions

The following conventions are used throughout this guide.

`⎕io←0`	All examples in this guide assume that index origin is set to `0`.
`⍺` and `⍵`	Two APL symbols are used in syntax descriptions of monadic and dyadic functions, *alpha* (`⍺`) identifies the left argument and *omega* (`⍵`) identifies the right argument.
abcde... *ABCDE....*	Examples of input to and output from SHARP APL are shown in the APL union character set, in which the first and second APL alphabets appear in lowercase and uppercase, respectively. For more information, see "The APL Union Character Set" below.
`constant width`	Examples of non-APL input and output (e.g., JCL) are represented in a `constant width` typeface.
passno	Argument variables appear in *sans-serif italic* type.
[ ]	Optional arguments are shown between square brackets. For example, in the following APL statement, the argument *passno* is optional:

'filename' ⎕stie tieno [,passno]

The APL Union Character Set

The names of APL files and workspaces may be constructed from the characters *a–z*, *A–Z*, and `0–9`, as well as the glyphs *underbar* (_), *delta* (`⍙`), and *delta-underbar* (`⍫`). The characters *a–z* are referred to as the first alphabet, and appear as lowercase letters in this guide. The characters *A–Z* are referred to as the second alphabet, and are represented as uppercase letters.

Due to the wide variety of input and output devices available for SHARP APL, these alphabets may display differently on your terminal. For example, the first alphabet is often displayed as uppercase, while the second alphabet is displayed as underscored uppercase. To determine what the first and second alphabets look like on your terminal, display the locations `⎕av[86+⍳26]` and `⎕av[113+⍳26]`.

APL workspace and file utility selection statements represent these characters as follows:

In APL Union	*In EBCDIC*
_ (underbar)	_ (underscore)
⍙ (delta)	- (minus)
⍫ (delta-underbar)	+ (plus)

The display of names is controlled by the `EXEC PARM` parameter `LOWERCASE=YES|NO`. If `LOWERCASE=YES` is specified, then the above convention is also used to write to `SYSPRINT`. If `LOWERCASE=NO` is specified, then second alphabet APL characters are displayed in uppercase, and a second line is printed, without carriage advance, underscoring these letters.

Related Publications

The following manuals are distributed with SHARP APL Version 23. The system administrator should ensure that these manuals are available.

SHARP APL for OS/390,

- *Interface to MVS JES Configuration, Operation and Maintenance Guide,* publication code MM-014-0302
- *Internal and Operational Changes,* publication code MM-000-1203
- *Messages* and *Codes,* publication code MM-000-1603
- *SVP Configuration, Operation and Maintenance Guide,* publication code MM-001-0302
- *System Administrator's Guide,* publication code MM-000-1003
- *System Maintenance Guide,* publication code MM-000-1103
- *Upgrade Guide,* publication code MM-000-0203
- *Utilities Manual,* publication code MM-000-1303

The following manuals represent the SHARP APL for OS/390 product extensions that are updated with Version 23:

SHARP APL for OS/390,

- *APLMON* User's *Guide,* publication code MM-006-0403
- *Batch Task User's Guide,* publication code MM-034-0401
- *Usage Inquiry System User's Guide,* publication code MM-031-0401

For a complete list of SHARP APL publications, please visit the Soliton Associates website:

`http://www.soliton.com`

Look for the "Publications" reference after choosing "Support" on the HOME page.

Contacting Soliton Associates

Problems or questions regarding your SHARP APL system or this guide should be directed to the Soliton Associates Technical Support group:

`zdist@soliton.com`

Requests for additional technical documentation should be directed to your account manager or the Soliton Associates Sales group:

`sales@soliton.com`

1 Introduction

Overview

SHARP APL for OS/390 Version 23 (V23) is an upgrade to SHARP APL for OS/390 Version 22.2. Changes to SHARP APL in V23 include modifications and additions to the base APL system, as well as some changes to the SHARP APL utilities and other related software products.

Summary of Changes

Language Enhancements

The following new and changed functions in the SHARP APL language are included in Version 23.

- Underbars are allowed in APL names.
- The Euro Currency symbol is now available.
- □*avm* has been added.
- A fault in □*io* has been corrected.
- System Errors in □*pp* have been corrected.
- A fault in *less* has been corrected.

Intrinsic Functions

The following new and changed Intrinsic Functions are included in Version 23.

- list of Intrinsic Functions available in the *sapl* group as of Version 23
- date conversion Intrinsics
- *sapl.nextprime*

- `sapl.rowmatch`
- `sapl.fbrowmatch`
- `sapl.translate`

Other Enhancements

- case insensitive passwords
- `)cc` corrections

2
Language Enhancements

Overview

The following new and changed functions in the SHARP APL language are included in SHARP APL for OS/390 for Version 23.

Enhancements

Allow underbars in APL names

Underbar (_) is now allowed in variables, functions, groups, packages, files, passwords (in sign-on, workspace locks), workspaces, and binds. Underbars are legal as a non-initial character. For example, "*the_name*", but not "*_name*". This provides compatibility with SHARP APL for UNIX and with the APL standard.

The Euro Currency symbol is now available

SHARP APL for OS/390 `⎕av[217]` contains the new Euro Currency Symbol. This corresponds with SHARP APL for UNIX `⎕av[178]`.

This change permits the euro symbol (`⎕av[⎕io+217]`) to be entered in input lines and included in code strings and functions within comments and character constants. It thus becomes a similar character to, for example, most National Use characters.

⎕*avm* Added

A new niladic system function ⎕*avm*, whose result is identical to that of ⎕*av*, has been added for compatibility with SHARP APL for UNIX code.

Changes

System Errors with Enclosed Arrays

If the display of an enclosed array required the value of □*pp*, the display could result in a system error. The display form of the array was corrupted and, when the display form of the array was marked garbage, so was the new value of □*pp*. The following sequence reproduces the problem:

```
□pp←1 ⋄ 0.1⊃ abc ⋄ □pp←1
```

This fault has been fixed.

□*io*

The share level of system variables was not updated in)*copy*. This has been corrected so that after a)*copy*, QUAD variables like *io*, *pp*, and *pw* are examined and validated.

Previous behavior:

```
      )clear
      □io
1
      ⍳5
1 2 3 4 5

      )copy somews □io
      □io
0
      ⍳5
1 2 3 4 5
```

less

The wrong answers and system errors in *less* or dyadic '~' with enclosed arguments caused by garbage collection when obtaining the result have been fixed. These tended only to happen when □*wa* was low.

Problems which were encountered with boolean and enclosed array arguments have been resolved.

The built-in rank operator support has been disabled. The built-in rank operator support in *less* returned wrong answers. *less* with the rank operator now uses the slower general rank operator support, but will return correct answers.

3
Intrinsic Functions

Overview

The following new and changed Intrinsic Functions are included in Version 23.

sapl. Group

The Intrinsic Functions available in the *sapl* group as of Version 23 are:

sapl.translate	translate character argument with supplied translate table
sapl.type	return type and structure of *rha*
sapl.rowmatch	return row-index in *lha* of matching *rha* rows
sapl.rowhash	return hash-index in *lha* of rows in *rha*
sapl.browmatch	return row-index in *lha* of matching *rha* rows
sapl.browhash	return hash-index in *lha* of rows in *rha*
sapl.sbrowhash	return row-index in *lha* of matching *rha* rows
sapl.sbrowhash	return hash-index in *lha* of rows in *rha*
sapl.fbrowhash	return row-index in *lha* of matching *rha* rows (floating point)
sapl.toymd	return the date from an argument giving the number of days since January 1st, 1600
sapl.Fromymd	return number of days since January 1st, 1600
sapl.cfa	return a character list which represents *rha*
sapl.afc	return the array represented by the linear rep
sapl.fsm	return an integer list which partitions the argument, and gives the resulting *fsm* state for each character.

sapl.prsr	use in combination with *sapl.fsm* to parse character text
sapl.mintype	return array coerced to the most compact data type possible, without data loss
sapl.nubsort	return nubsieve of a sorted character or numeric array of rank 2 or greater
sapl.pplusred	return partitioned plus reduction of *rha*
sapl.nextprime	find lowest prime equal to or larger than starting integer

Enhancements

Date Conversion Intrinsics

sapl.toymd

r ← *toymd* ***argument***

return the date from an ***argument*** giving the number of days since January 1st, 1600

- ***argument*** is any non-negative integer array
- ***result*** is an array whose shape matches that of the ***argument***, plus a final dimension of 3

  The result will be a normalized date (i.e., the month will be an integer between 1 and 12; the day will be a day number within that month).

Examples:

```
      toymd 0 145869
1600 1  1
1999 5 18
```

To compute the date 30 days hence:

```
      toymd 30+fromymd 3↑⎕ts
```

sapl.fromymd

r ← *fromymd* ***argument***

return number of days since January 1st, 1600

- ***argument*** is any integer array whose final dimension is 3

- *result* is an array whose shape matches that of the *argument*, less the final dimension of 3

  The result will be non-negative integers, the number of days from January 1st, 1600 to the *argument* date.

Example1:

```
      fromymd 2 3P1600 1 1 1999 5 18
0 145869
```

To compute if two dates are the same day of the week:

```
      0=7|(fromymd date1)-fromymd date2
```

Note: It is not required that the date be normalized, i.e., that the month number be from 1 to 12, and the day number correspond to a day within that month. Thus the function can be used (in conjunction with its inverse, `toymd`) to detect badly formatted dates (since `toymd` does produce normalized dates).

sapl.nextprime

`r ← sapl.nextprime` *na*

Set `r` to the next prime(s) equal to or greater than *na*.

na may have any shape, but must be a simple numeric array. None of the elements can have a magnitude greater than or equal to 2*31, and each of the elements must be tolerantly equal to the integer of nearest value.

The result will have the same shape as *na*. For each element (e) of *na*, the corresponding element of the result will be e if it is prime. Otherwise it will be the (positive) prime nearest to but greater than e. If e is less than 2, the result will be 2.

sapl.fbrowmatch

`r ←` *matrix* `fbrowmatch` *argument*

return row-index in *matrix* of matching *argument* rows
(floating point, binary-search, `rowmatch`)

This Intrinsic performs the services of `sbrowmatch`, but on floating point data, without fuzzed conversion to integer. The function is not an extension of `sbrowmatch` but does coincide with it when the data is pure integers. The left argument is presumed to be in upgrade order. All comparisons are done without fuzz (i.e., as if comparison tolerance were zero).

- *argument* is any simple character or real-valued vector or higher-rank array
- *matrix* is any simple character or real-valued matrix or higher-rank array. It is strongly assumed (but not verified) that the left argument is sorted in ascending order.

If the left argument is not in sorted order, the results will be virtually meaningless.

- *result* is a numeric vector whose result is the same as *browmatch* for any row which is found in the left argument. That is, it is the 0-origin index of the first row of the left argument which matches the (possibly padded) right argument row.

  If the row is not found, the result is:
  `-(1+`the index of the first row of the left argument which is greater than the right row`)`

  fbrowmatch differs from *browmatch*, *rowmatch*, *browhash*, and *rowhash* in that it treats real arguments in arithmetic rather than logical order. The APL *upgrade* function ⍋ may be used to create character or real left arguments to *fbrowmatch*.

  The "not-found" elements of the *fbrowmatch* result may be used to insert the not-found rows into the left argument.

For integer data and for floating point data coercible to integer without loss of precision, *fbrowmatch* gives the same result as *sbrowmatch*. For floating point data which is coercible to integer, but with loss of precision, *fbrowmatch* gives answers different from *sbrowmatch*. For floating point data which is not coercible to integer at all, *fbrowmatch* gives answers where *sbrowmatch* gives *domain error*.

Changes

sapl.rowmatch

Most SHARP APL primitives permit comparison of character and numeric data, returning `0`. As originally specified, the *rowmatch* intrinsic function signaled a *domain error* if one argument was character and the other was numeric, even if one or both arguments are empty.

In response to customer requests, we are changing this specification to return a result if one or both arguments are empty, rather than signal a *domain error*. If the right argument is empty, an empty result is returned; if the right argument is non-empty, an appropriately shaped "not found" (`¯1`) result is returned.

domain error will continue to be signaled in cases where neither argument is empty.

sapl.translate

There was a fault in *sapl.translate* whereby the left argument was ignored if the 256 residue of its address was zero. The *translate* intrinsic would fail to perform any translation and returned its right argument without applying the required translate table. This has been corrected.

4
Other Enhancements

Enhancements

Case Insensitive Passwords

When a task signs on to the SHARP APL system, the password is tried twice – as is, and reversing the alphabet. This simplifies the logon process when the translation table or the emulator used reverses the alphabet.

To avoid confusion during sign-on, especially when signing-on from terminals with different tables, the 3rd alphabet (case) characters typed in old and new password at sign-on have been altered:

- If only 3rd case characters are typed, convert them all to 1st case.
- If 2 cases are typed, convert all 3rd case characters to the case not typed.
- If 3 cases are typed, make no changes (must have been a typing error).

This password "folding" processing precedes existing code which retries each failing password, after changing all 1st case characters to 2nd case and vice versa.

This change does have the effect of accepting passwords that were previously rejected.

)*cc* Corrections

`37I` has been changed to accept a *keyw/dict* transliteration option for 3rd/8bit *cc* command. Previously, it was available only for new/normal *cc* command.

Whenever a user enters)*cc* *3rd* (or via the *Cc* function) on an ASCII type device, APL switches to allow 8 bit □*arbin* input instead of normal 7 bit input.

An option has been added to `6I`:

`6I15 0`	default - translated □*arbin/out*
`6I15 1`	enable S-task to do untranslated (8bit) □*arbin/out*

The per user setting for `)cc 3rd` was set, but not recognized at the sign-on time. It is now possible to save this setting per user number.

SVP

□*svo* has been enhanced to support ~ (tilde) as a marker (a prefix on either the variable name or the surrogate name) signifying a desire for SVP "no reconnect" processing for the share. If the marker is specified, a new result is possible: `¯1`. A `¯1` result means that no-reconnect was specified when the share was created, and the share has been coupled and retracted by the other side, and thus can no longer couple under any circumstances.